Assignment 5: Sorting Lists

CS 301

February 27, 2018

In this assignment we will look at several sorting algorithms for lists: Insertion Sort, Bubble Sort, Selection Sort, and Insertion Sort. These four sorting algorithms are described below.

Insertion Sort

- This may be how you are used to sorting things.
- We start with a list of unsorted items, and an empty list of sorted items.
- At each step, we take one item from the unsorted list, and move it into the sorted list in the spot it goes, by looking through the sorted list from the beginning until we find the correct spot to insert it.

Bubble Sort

- We are given an unsorted list.
- We run through the list multiple times.
- On each run, we compare all pairs of adjacent elements (first, the elements at indices 0 and 1, then 1 and 2, then 2 and 3, *etc*.), and switch them if they are out of order.
- After each run, at least one more element will have "bubbled" up to its correct place at the end of the list. (Be sure you see why.)
- We keep doing runs until the whole list is in order.

Selection Sort

- Recall that at the end of each run in Bubble sort, one more element will be at its correct spot at the end.
- This suggests an alternate strategy: on the ith run, just look for the ith biggest element, which will be the biggest element that we haven't put into the correct spot yet, and swap it into the spot it goes, which will be the ith spot from the end.

Merge Sort

- First, write a function that inputs two sorted list and merges them to create one sorted output list containing all the items from both lists.
- Note that you can do this by moving up from the bottom of each list and comparing the smallest item left in each list at each point to determine the next item that goes into the merged list.
- Now, write a recursive function to sort a single unsorted list by dividing it in half, recursively sorting each half using Merge Sort, and then merging the two sorted halves together.

1. Experiment in groups or on your own with each of these sorting algorithms until you can consistently sort a list of numbers by any one of these methods.
2. Write Python functions to implement each of these sorting algorithms.
3. For each function, write testing code to make sure it works correctly.
4. For each function, look at your working code, and try to determine the worst case Big Oh running time of the code. Explain what it is in your comments.
5. You can compare your code to that given in the textbook sections 6.6, 6.7, 6.8, 6.9, and 6.11. If you've followed the instructions above, two of your functions will be somewhat different from what is described in the textbook.